Carbon Charts: Development Process

McMaster University
Group 18

Darsi Anandarajah
Justin Licari
Eliad Moosavi
Thomas Mullen
Richard Zhang

November 30, 2018

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**	**Revision**
30/11/2018	1.0	Created document	REV-0

1 Introduction

This document outlines system and component. Additionally,this document will serve as a complete declaration of the overall process-workflow to be used in the development and implementation of Carbon Charts.

2 Terminology

1. Agile - Agile software development is a group of software development methodologies based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams.

2. Workflow - The sequence of processes through which a piece of work passes from initiation to completion.

3. VCS - Version Control System

3 Version Control

Carbon Charts is an open source charting library that will serve as a functional component of IBM's Carbon Design System. Its open-source nature requires that source code must not only be subject to version control just as any software project should but, also be publicly available. Git is used as the VCS for tracking changes in computer files and coordinating work on those files among multiple people. The library is publicly hosted as an open-source project on github, a hosting service for Git repositories. Please note, git is the VCS, the tool, while github is the service for products that use git. The benefits of git include:

- Branching and Merging: Provides frictionless context switching, role-based code lines, feature based workflow and disposable experimentation.

- Strong support for non-linear development: Git supports rapid branching and merging and includes specific tools for visualizing and navigating a non-linear development history.

- Fast and Lightweight

- Distributed: Aimed for speed, multiple backups, any or endless number of workflows, subverion-style workflows and integration manager workflows.
- Data Assurance: Ensures cryptographic integrity of every bit in a project.
- Staging Area: This is an intermediate area where commits can be formatted and reviewed before completing the commit.
- Free and open-source

4 Process Workflow Overview

This project employs an agile development framework consisting sprints, no longer than two weeks. Outlined below are steps of the overall workflow from task creation to completion.

4.1 Task Creation

This step consists of generating new tasks by examining the projects requirements via team discussion. Tasks involving the implementation of new features or elements are first approved by IBM's design team before implementation. All tasks will be considered as long as they are valid and not contrary to the requirements or permanently out-of-scope for the project. Task are prioritized in the next step.

Tasks will be created and tracked using Github Issues and will be labeled "feature", "bug", or "development" as appropriate. Features are software items with distinguishable characteristics that add performance, portability, or functionality to the existing library. Bugs are issues in the current iteration or version of the software. Development tasks are tasks that do not include direct changes to the software, but instead require a change to build process which includes building, testing, or deployment tools.

4.2 Task Prioritization

In this step, tasks created in the previous step will be assigned a priority. The available priorities are ”Low”, ”Medium”, ”High” and ”Backlog”. The backlog priority implies the feature may be indefinitely delayed without any significant detriment to the project.

For features, this priority is determined by the necessity of the feature to satisfy the requirements. It must also be considered that a feature may ”block” another feature, in which case the feature inherits the highest priority of any blocked feature.

For bugs, the priority assigned is determined by the severity of the issue. High severity bugs prevent the software from functioning at all, medium severity implies severe degradation of utility, and low severity implies a aesthetic or avoidable problem.

4.3 Task Assignment

This step consists of assigning tasks to a team member who becomes the owner of that task. Ownership may be transferred or shared with agreement of both members. Assignment will be decided by workload and competence. Only ”backlog” tasks may be unassigned. The owner of the task will be assigned the task on Github Issues and is responsible for its completion.

4.4 Task Implementation

This step involves the implementation of the task and writing code, consistent with contributing and code-style guidelines as well as strict adherence to correct programming principles.

Development will only require Git and Node.js with a minimum version of 9.0.0. All other tooling and dependencies can be installed by running ”npm install” after cloning the repository. This creates a low barrier to entry for new developers.

When part of a task is completed, it should be committed using Git to a branch dedicated to a specific task. Commits must be made using the tool Commitizen which enforces semantic versioning and descriptive commit messages.

When the entire task is completed, a Github pull request should be made with an explanation of all changes made. The following validation step must then be completed.

4.5 Task Validation

For a task to be considered complete, it should pass a series of validation steps.

The first step consists of an automated continuous integration test suite that runs on every pull request. This will catch many regressions, code style issues, and build failures without requiring any human effort. Contributors are expected to add additional tests for their new code, which will also be run automatically.

Netlify is used to deploy component demos for every commit in a pull request. As new commits are added to a pull request, Netlify will add a comment inside the pull request with a link to the deployed demos.

Continuous integration will use IBM's Typescript linter to ensure a consistent code style and the Jasmine testing framework to run unit tests and code coverage. Travis CI is used to perform continuous integration operations. Integration of CI with Github results in a status bar shown in every pull request with health statuses on unit tests and lint reports.

The second step is code review by a second member. The goal of this review is to ensure the code actually implements the intended feature. A code style review should also be done, ensuring architectural patterns are being followed and that code is clear and readable.

When all commits are validated, it is the task owner's responsibility to resolve any merge conflicts and request re-validation if anything may have broken during conflict resolution. The code is then merged and the task completed.

5 References

1. Parnas, D.L. ”Designing Software for Ease of Extension and Contraction”, Proceedings of the Third International Conference on Software Engineering, pp. 264-277, 10-12 May, l978.

2. https://opensource.guide/best-practices/

3. https://git-scm.com/about/branching-and-merging

4. https://www.netlify.com/docs/continuous-deployment/